Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Argentina

Department of Computer Science

School of Sciences

University of Buenos Aires

Argentina

Comparison of HP and Intel's mathematical libraries and the implementation of parallel solvers for engineering applications on Itanium platforms

Martín Degrati, Esteban Franqueiro, Ximena Pisani, Xavier Ruíz Ibañez and Hugo Scolnik

Abstract

Our research is focused on developing highly efficient parallelizable solvers for huge systems of linear equations that arise from finite element discretization of complex nonlinear engineering problems. Those problems are nonlinear, require many linearizations, and hence several days of CPU time on Itanium platforms. Another important application is the reconstruction of tomographic images. This work includes a comparison of Intel's MKL and HP's MLIB mathematical libraries on Linux, from the point of view of the performance on numerical problems using sequential and parallel implementations. The new solver uses Level 1, 2 and 3 BLAS double precision routines and functions.

Part One

The Linear Solver Architecture and the Mathematical Libraries

Problem Statement

The problem of solving very large asymmetric systems of linear algebraic equations appears in many real world applications like the reconstruction of tomographic images, in computational mechanics, etc.

Beyond certain sizes, direct methods can not be applied. Due to that reason iterative methods are employed, and a large variety of algorithms have been developed. Those based on the so called Krylov subspaces are very popular (like GMRES, BICG, etc), all of them requiring preconditioners.

Their main limitation is that convergence can not be guaranteed, and this is a severe shortcoming for many applications like PET (Positron Emission Tomography), or in Engineering where difficult problems arising from the use of the Finite Elements Method may require several days of CPU time.

In the last years we have developed a new category of iterative methods that share very significant advantages: **they are**

1. **Globally convergent**
2. **Fast**
3. **Intrinsically parallel**

Research Direction

- ***Parallel Linear Solver***: The use of iterative linear solvers allows efficient numerical computations in problems where direct library solvers cannot be used due to the increasing demands of CPU power and memory of the problems being solved.

- ***Numerical Libraries***: Basic linear algebra operations used in the solvers' implementation are needed to be accurate and efficient. This requires some effort in the selection and optimization of the basic library routines.

Parallel Linear Solver

The iterative parallel solver used is an implementation of the ACCIM algorithm. This algorithm has some advantages:

- It’s an iterative solver. An implementation of this kind of solver has better accuracy and lower system requirements than classical methods (LU, QR, etc).
- It’s a highly parallelizable method. The set of system equations can be easily partitioned in an arbitrary number of disjoint sets of equations, and those sets can be processed in parallel.

Numerical Libraries

Our research also covers a performance comparison of Intel's MKL and HP's MLIB mathematical libraries on Linux operating systems using sequential and parallel implementations. This comparison includes tests on:

- Performance of the BLAS routines at Levels 1, 2, 3 excluding complex data types.
- A comparison of the new implementation of the ACCIM algorithm and the classical methods are also included.

Parallel Solver Architecture

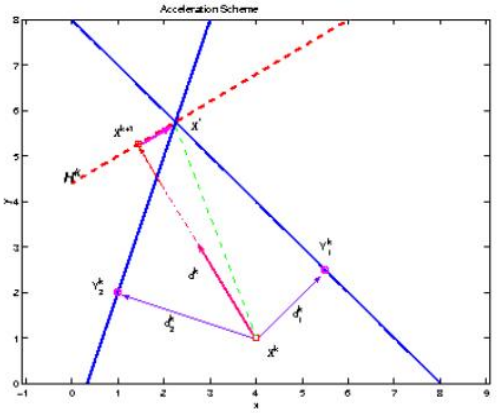
FEM Aplications

Tomographic Hardware

On-Demand Numerical Computing Services

Linear System Solver

ACCIM Algorithm



7. **while** $r > \varepsilon \times \max(1, r_0)$
   - 7.1. $k = k + 1$
   - 7.2. $x^k = x^{k-1} + \lambda_{k-1} d^{k-1}$
   - 7.3. **for** $j = 1...N$ (in parallel)
     - 7.3.1. **for** $i = b^j ... e^j$
       - 7.3.1.1. $d_i^k = r_i^k \times \frac{a_i}{\|a_i\|} = \left(\frac{b_i}{\|a_i\|} - \frac{a_i}{\|a_i\|} \times x^k \right) \times \frac{a_i}{\|a_i\|}$
       - 7.3.1.2. $D_k^j = \sum_{i=b^j}^{e^j} d_i^k$
       - 7.3.1.3. $R_k^j = \sum_{i=b^j}^{e^j} \left(r_i^k \right)^2$
   - 7.4. $d_k = \sum_{i=1}^{N} D_k^j$
   - 7.5. $\alpha_k = \frac{\left(d^{k-1} \right)^t d^k}{\left\| d^{k-1} \right\|^2}$

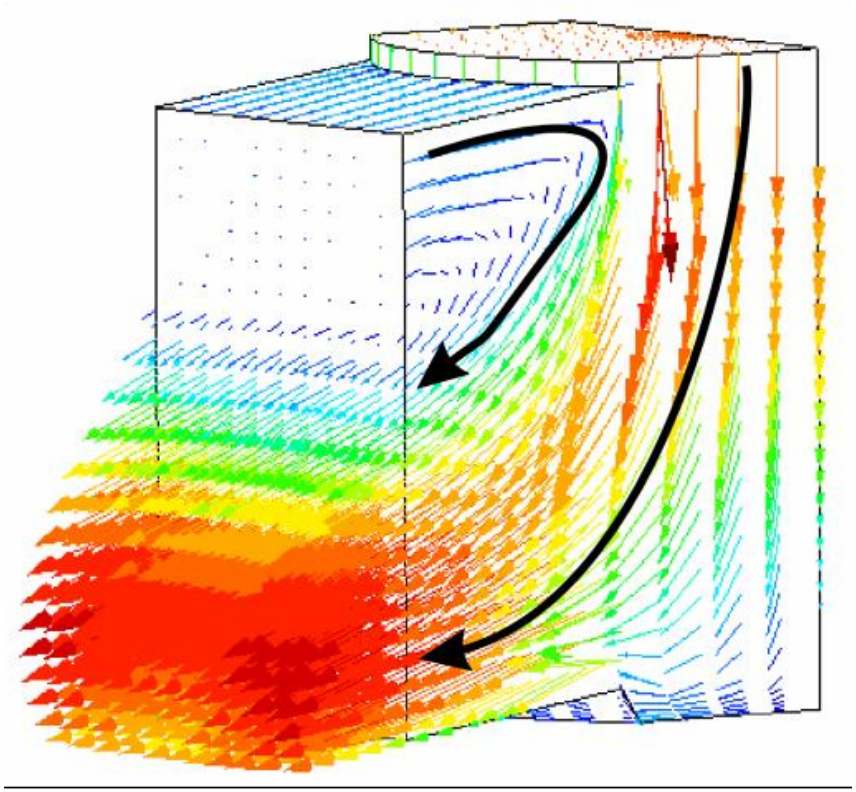
Intel MKL $\alpha_k = \frac{\left(d^{k-1} \right)^t d^k}{\left\| d^{k-1} \right\|^2}$

HP´s MLIB $\lambda_k = \frac{r_k}{\left\| d^k \right\|^2}$

Part Two

Applications

Pipe Piercing Modeling



The production of forged pipes at Tenaris-Siderca needs to be modeled to increase the quality of the pipes, and to optimize the time process of the forge. The currently used Finite Elements Method program's time execution can be reduced by fast and robust linear equations solvers.

Finite element discretization of industrial problems

Industrial problems are characterized by:

- **Complex geometries**
- **Non-linear response**
- **Abrupt change of properties when interfaces are considered**

Hence, the resulting finite element models are characterized by:

- **Very refined 3D discretizations (great number of small elements)**
- **Iterative procedures in order to linearize the problem**
- **Important variations of properties in the model domain**

Finally, the linear systems that arise from those problems:

- **Have a great number of unknowns ($>10^5$) and sparse structure**
- **Must be solved many times due to the non-linear problem iterations**
- **are very ill-conditioned**

Requirements for the linear equation solvers

DIRECT SOLVERS

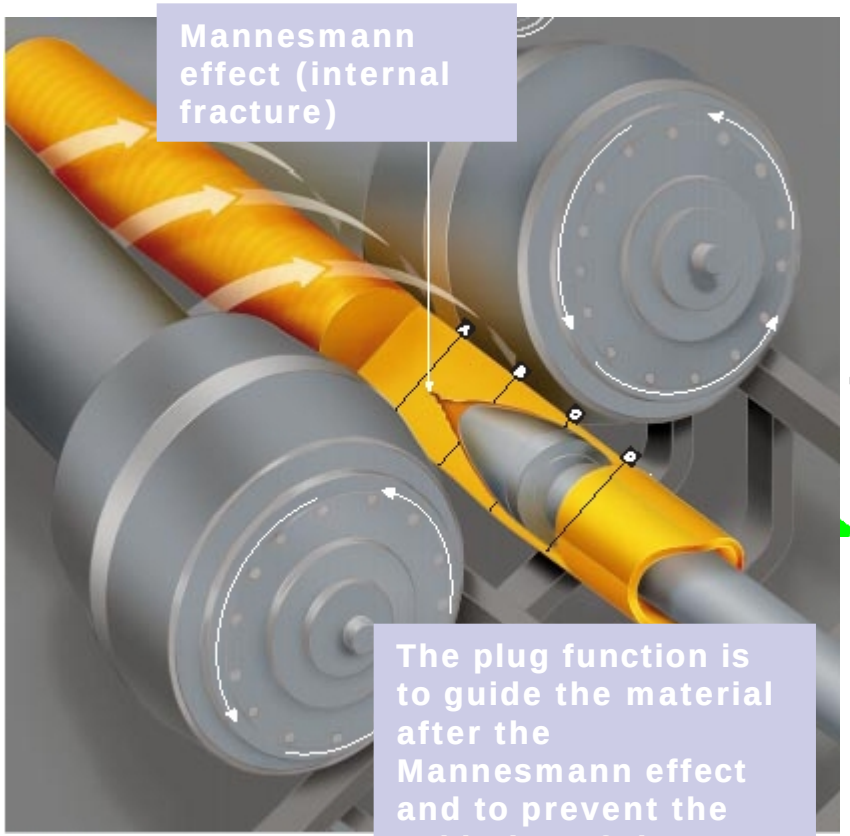
- Sparse implementation ⇐ **Allows solving great number of unknowns**
- Difficult parallelization
- Limited by memory requirements and round-off error propagation

ITERATIVE SOLVERS

- Easy parallelization
- Convergence difficulties in ill-conditioned systems ⇐ **Preconditioning techniques are required**

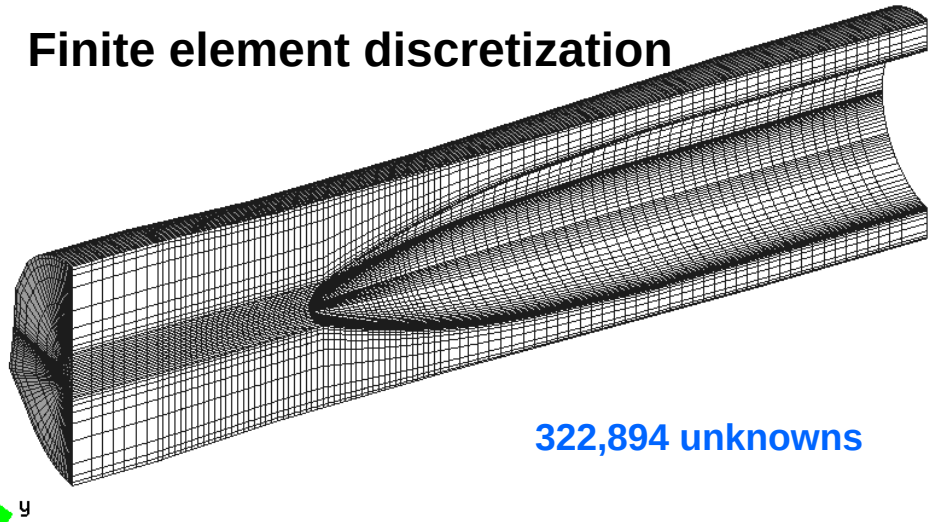
A PARALLELIZABLE ITERATIVE SOLVER CAPABLE OF DEALING WITH VERY ILL-CONDITIONED PROBLEMS WITHOUT AD-HOC PRECONDITIONING

Examples of engineering application of the finite element method: Simulation of the Mannesmann piercing process



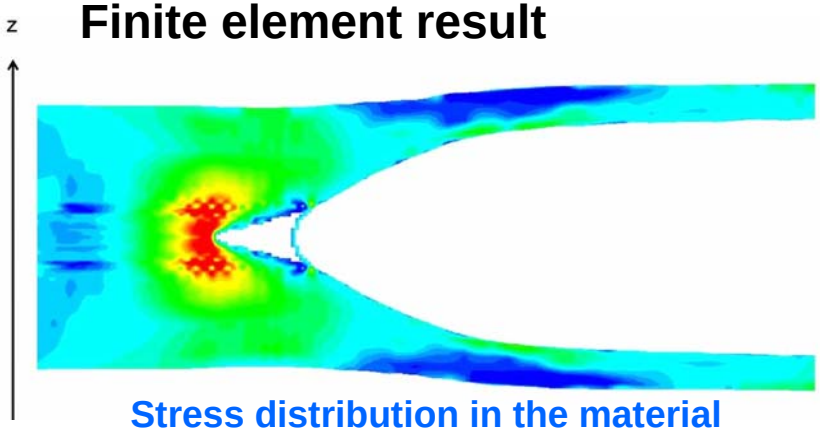
Industrial process

Finite element discretization



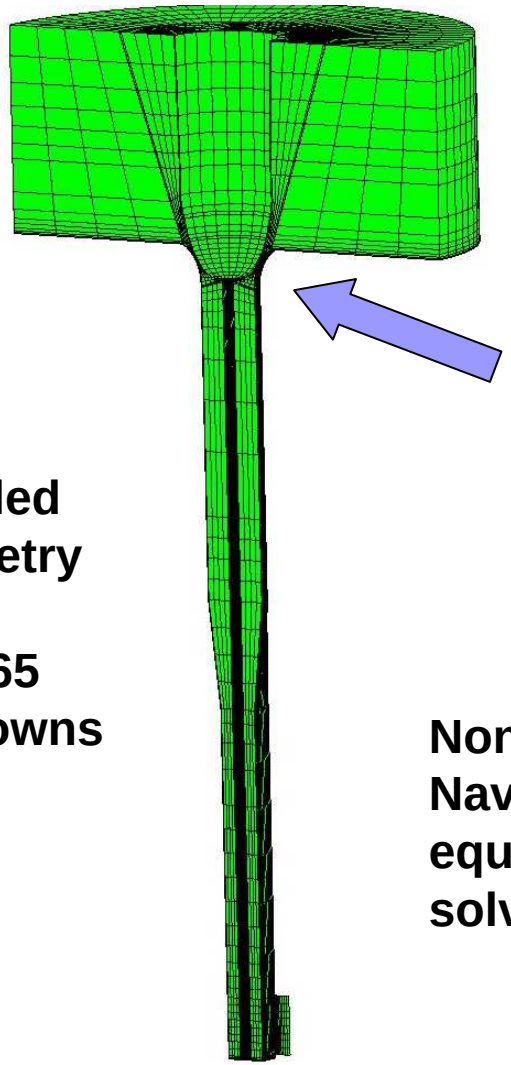
322,894 unknowns

Finite element result



Stress distribution in the material

Examples of engineering application of the finite element method: Control flow devices in the steel industry



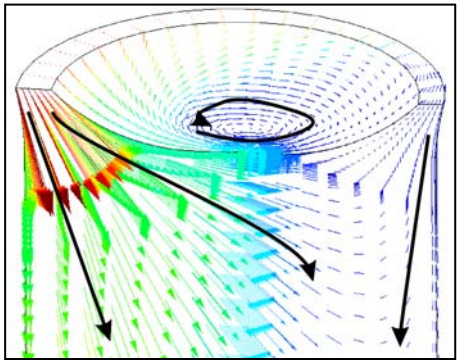
Modeled geometry

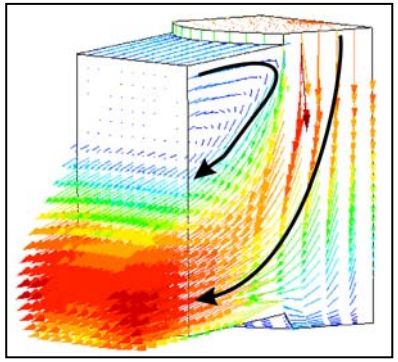
235,565 unknowns

Control flow device

Non-linear Navier-Stokes equation is solved

Model results





Velocities around the control flow device

Examples of engineering application of the finite element method:

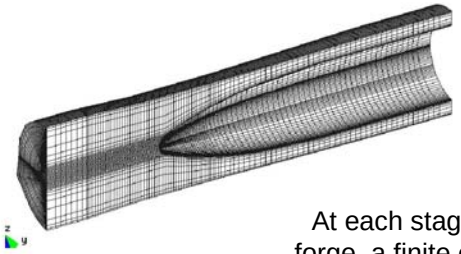
Linear solver information

Sparse direct solver HP MLIB v8.3 (64-bit implementation)

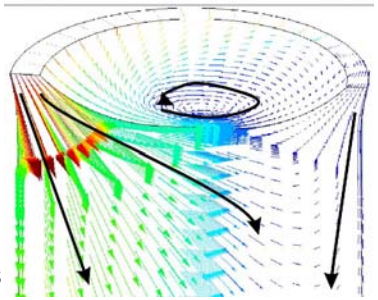
Engineering problem simulated	Memory used	Factorization [CPU time]
Mannesmann piercing process	4 GB	438s
Control flow devices	3 GB	114s

Thousands of factorizations are required for the convergence of the non-linear problem (more than 3-5 days of CPU time)

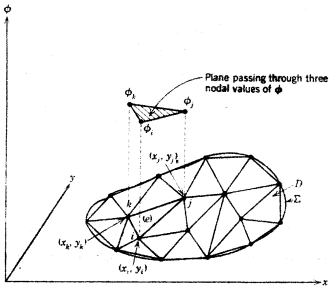
Pipe Piercing Modeling Process



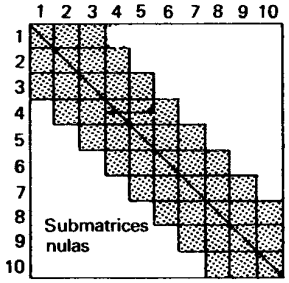
At each stage of the pipe forge, a finite element mesh is obtained



The stress/strain state at each point of the pipe permits the pipe-forge process modelling



A set of finite element equations are obtained according to the Galerkin´s method



The variational minimization is performed by solving, at each point of the mesh, a succession of linear equations system

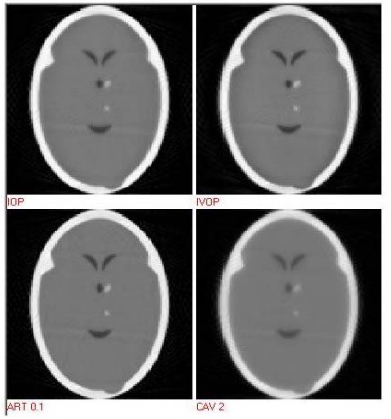
Each large linear system obtained is divided in blocks and solved in parallel in a Linux Itanium based server or cluster

Positron Emission Tomography

Large linear equations systems are needed to be solved fast and efficiently, due to increasingly resolution and time demands of the medical area

PET Images Processing

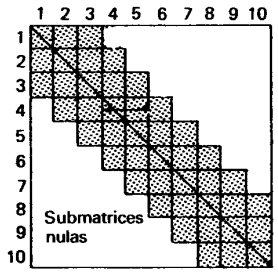
A PET tomograph is used to collect X-ray data from different angles through the body scanned



The brain image can be reconstructed and displayed

$$[S^*]_i = \begin{bmatrix} [S^*_u] & & \\ & [S^*_r] & \\ \text{Las matrices secundarias que no se muestran son cero} & & [S^*_\theta] \end{bmatrix}_i$$

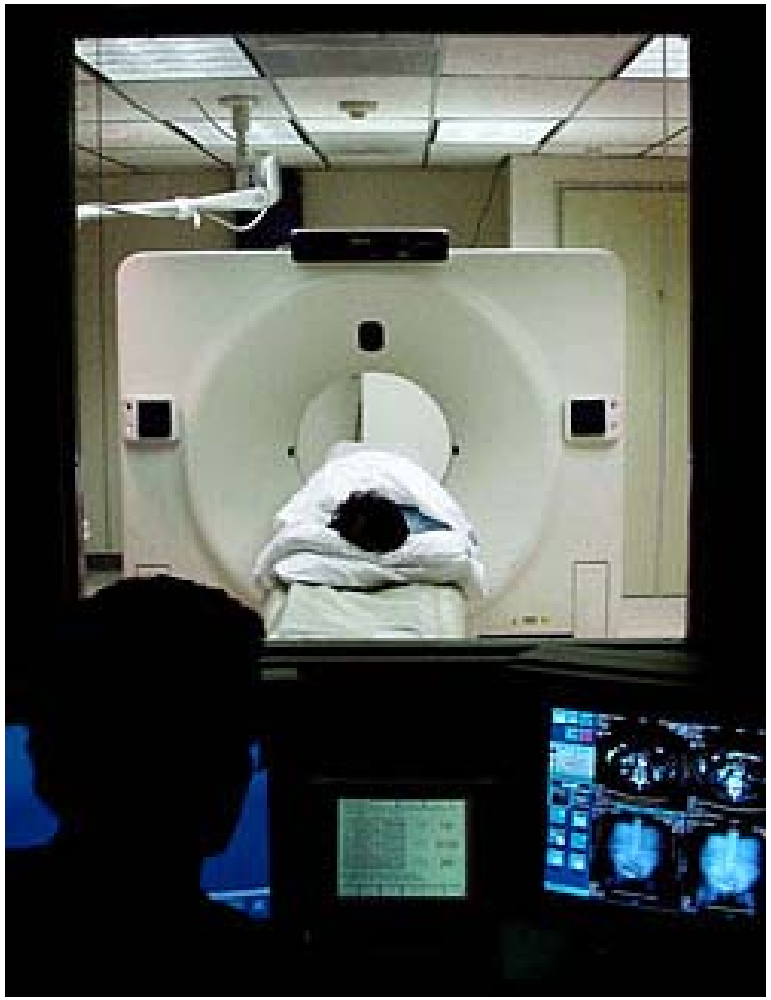
On each direction scanned, the measured attenuation of the X-ray can be used to construct a linear equation. The entire set of directions scanned results on a large linear equations system



The huge and sparse linear system is inconsistent and therefore a least squares solution must be computed

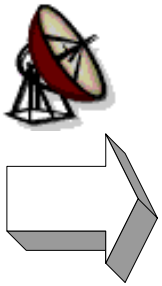
The large linear system obtained from LMS is divided in blocks and solved in parallel in a Linux Itanium based server or cluster

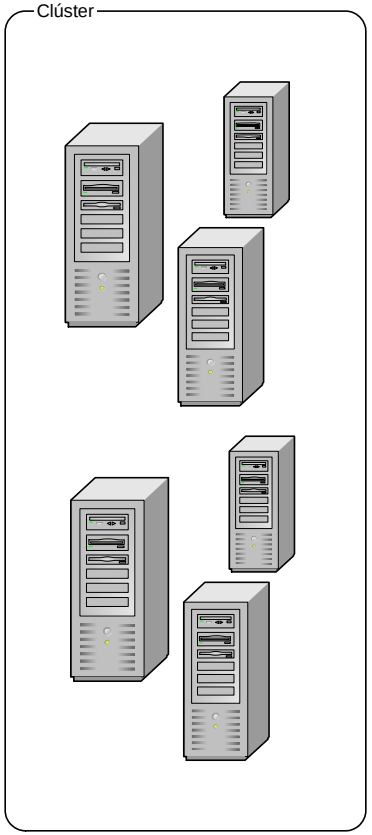
On-Demand PET Images

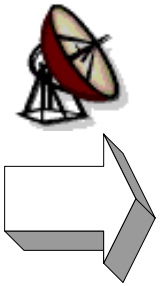


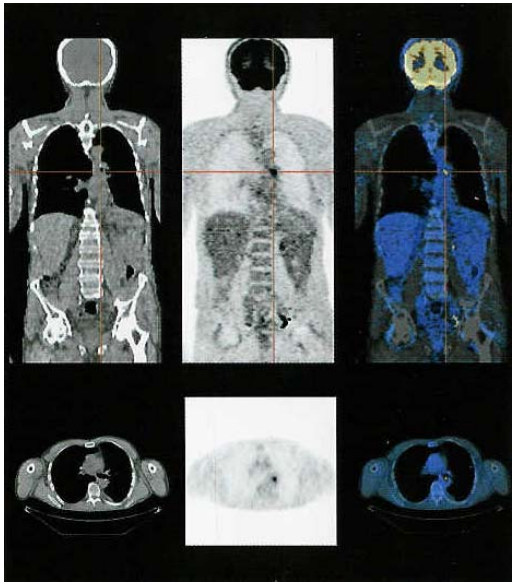
The huge amount of CPU power needed for tomographic image reconstruction can be translated from the tomographic hardware to a Remote Computer Center, in order to decrease medical hardware costs and improve time and resolution requirements on the image.

On-Demand PET Images Process









A PET tomograph is used to collect X-ray data from different angles through the body scanned. This raw data is transmitted to the computer center

The raw data received fom the PET tomograph is converted into a linear system, partitioned in blocks of equations, and finally solved in parallel by a Linux Itanium based cluster or grid

The reconstructed image is sent to the end users (doctors, hospitals, etc)

Part Three

MKL and MLIB BLAS performance comparison.

Routines and Functions tested

We tested all double precision routines and functions in Level 1, 2 and 3 BLAS on increasing vector dimensions.

Level 1 BLAS	Level 2 BLAS	Level 3 BLAS
dasum daxpy dcopy ddot dsdot dnrm2 drot drotm dscal dswap idamax idamin	dgbmv dgemv dger dsbmv dspmv dspr dspr2 dsymv dsyr dsyr2 dtbmv dtbsv dtpmv dtpsv dtrmv dtrsv	dgemm dsymm dsyrk dsyr2k dtrmm dtrsm

Test environment description

Hardware

Processors: 2x Intel Itanium 2

Clock frequency: 1.5 GHz

Cache: 6 MB L3

System bus bandwidth: 6.4 GB/s

Memory: 2 GB DDR, 8.5 GB/s bandwidth

Software

Operating System: Debian Linux Version 3.1r1 (Sarge)

Compiler: Intel C++ Compiler for Itanium-based applications Version 9.0.030 (20051201)

Math Libraries:

- Intel Math Kernel Library 8.0.1.006
- HP MLIB 9.2

Compiler switches used

For Intel MKL tests programs

```
icc -O3 -fno-alias -tpp2 -mcpu=itanium2 -IPF_fma -parallel -openmp
   -lpthread -D_USE_MKL_
   -I/opt/intel/mkl/8.0.1/include
   -L/opt/intel/mkl/8.0.1/lib/64
   -lmkl blas1.cpp -o blas1.out
```

For HP MLIB tests programs

```
icc -O3 -fno-alias -tpp2 -mcpu=itanium2 -IPF_fma -parallel -openmp
   -lpthread -D_USE_MLIB_
   -I/opt/mlib/intel_8.0/hpmpi_2.1/include
   -L/opt/mlib/intel_8.0/hpmpi_2.1/lib/64
   /opt/mlib/intel_8.0/hpmpi_2.1/lib/64/libveclib.so.1
   -lm -lunwind blas1.cpp -o blas1.out
```

Threading

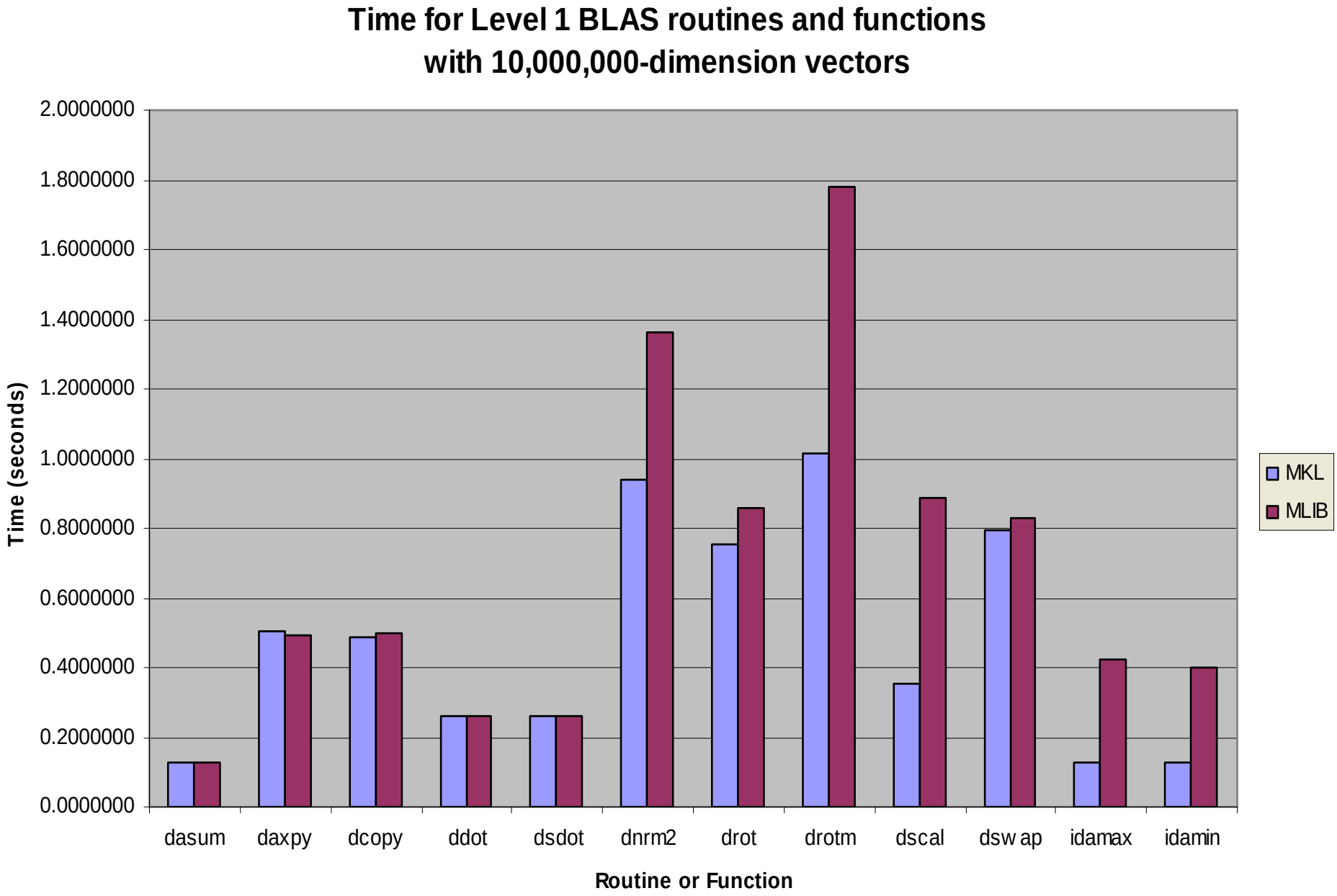
For Level 1 and Level 2 BLAS tests programs

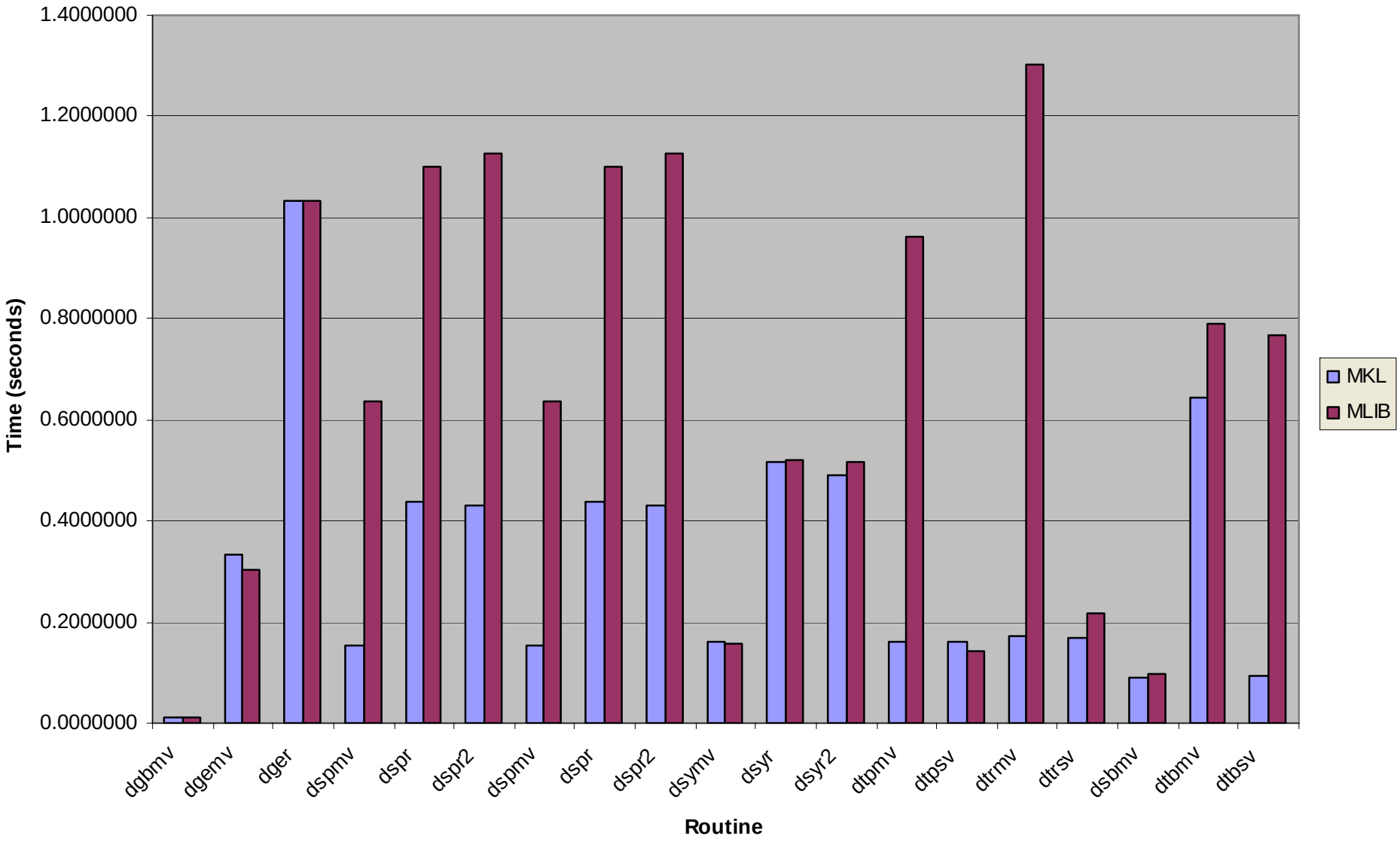
The programs were executed using the **OMP_NUM_THREADS** (for MKL) and **MLIB_NUMBER_OF_THREADS** (for MLIB) environment variables set to 1, as Level 1 and Level 2 BLAS routines and functions are not parallelized.

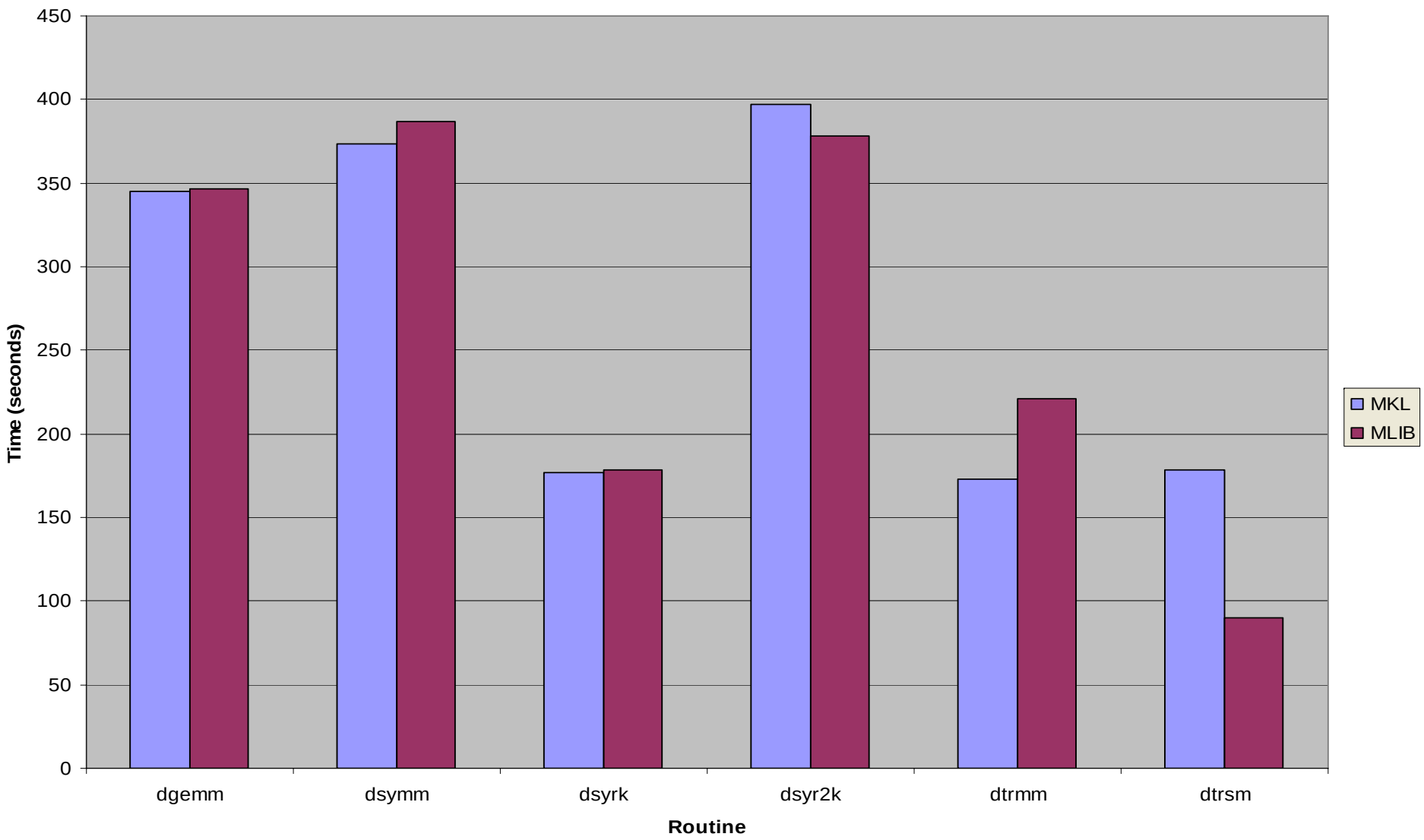
For Level 3 BLAS tests programs

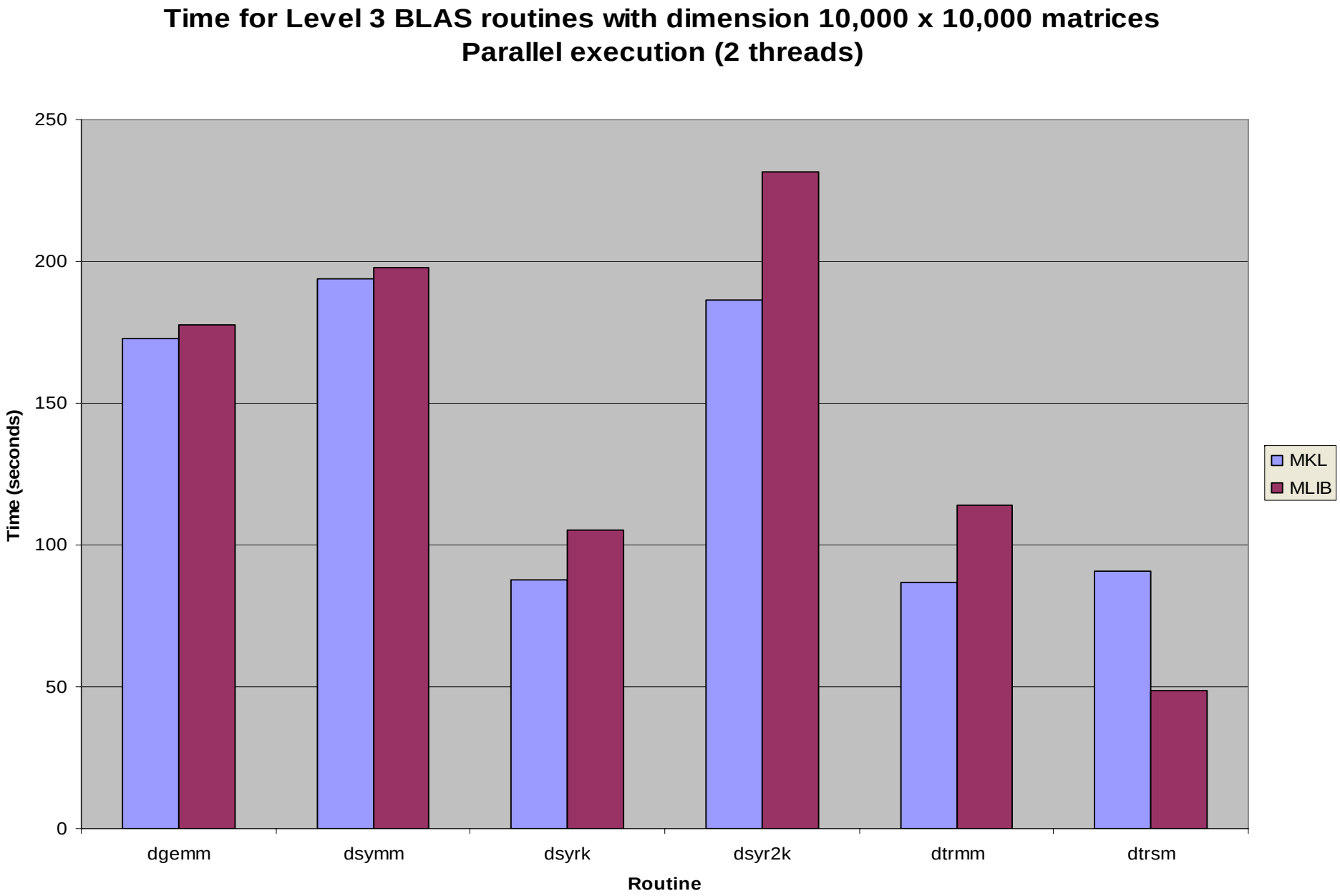
Level 3 BLAS routines are parallelized, so we set the values of the environment variables OMP_NUM_THREADS and MLIB_NUMBER_OF_THREADS according to the level of parallelism being tested.

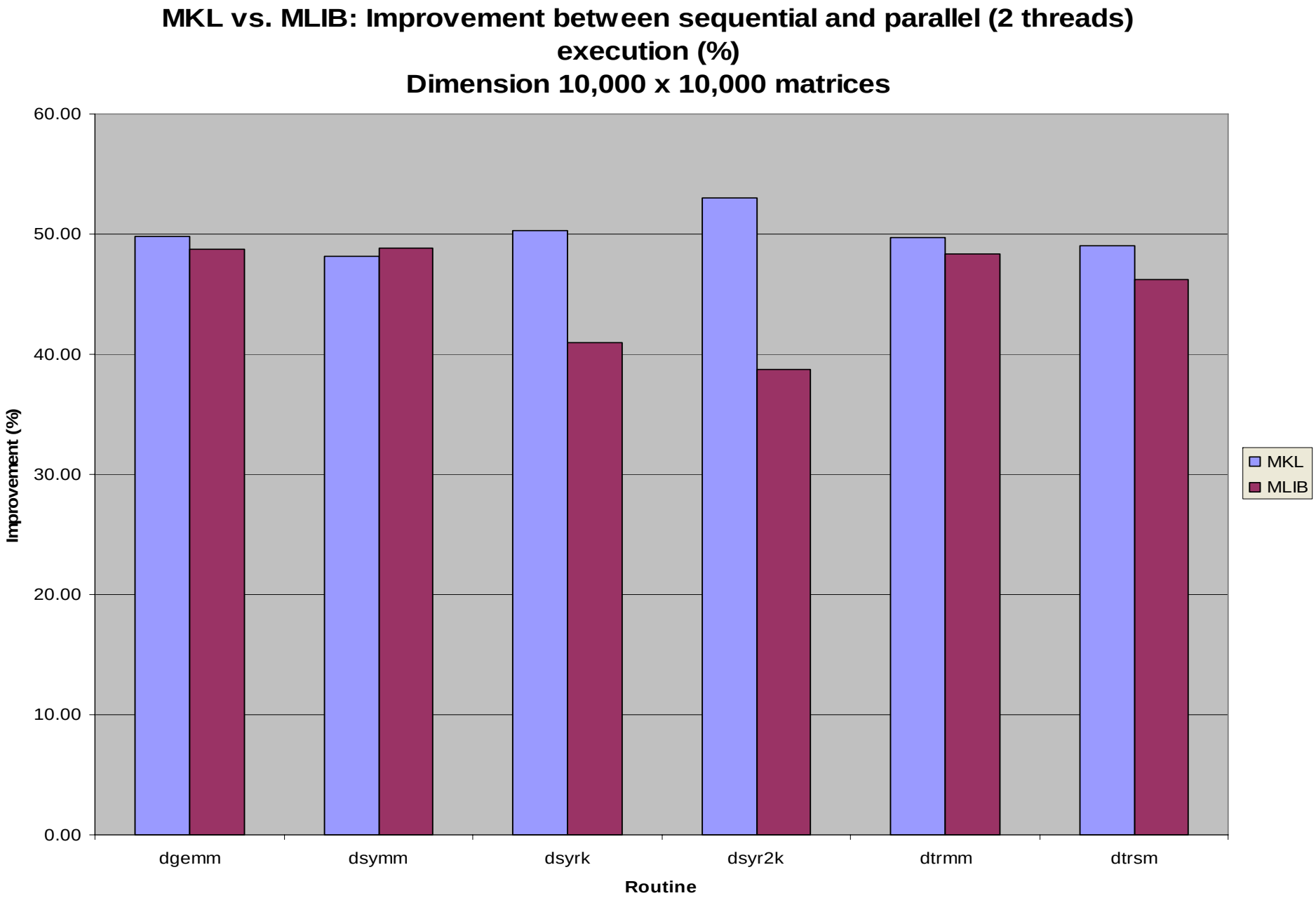
As we have two processors in our system, we have executed them with one and two threads.


Time for Level 1 BLAS routines and functions
with 10,000,000-dimension vectors
Time (seconds)
2.0000000
1.8000000
1.6000000
1.4000000
1.2000000
1.0000000
0.8000000
0.6000000
0.4000000
0.2000000
0.0000000
dasum
daxpy
dcopy
ddot
dsdot
dnrm2
drot
drotm
dscal
dsw ap
idamax
idamin
Routine or Function
MKL
MLIB


Time for Level 2 BLAS routines
with 15,000 x 15,000 matrices
Time (seconds)
1.4000000
1.2000000
1.0000000
0.8000000
0.6000000
0.4000000
0.2000000
0.0000000
dgbmv
dgemv
dger
dspmv
dspr
dspr2
dspmv
dspr
dspr2
dsymv
dsyr
dsyr2
dtpmv
dtpsv
dtrmv
dtrsv
dsbmv
dtbmv
dtbsv
Routine
MKL
MLIB


Time for Level 3 BLAS routines
with dimension 10,000 x 10,000 matrices
Sequential execution
Time (seconds)
450
400
350
300
250
200
150
100
50
0
dgemm
dsymm
dsyrk
dsyr2k
dtrmm
dtrsm
Routine
MKL
MLIB


Time for Level 3 BLAS routines with dimension 10,000 x 10,000 matrices
Parallel execution (2 threads)
Time (seconds)
250
200
150
100
50
0
dgemm
dsymm
dsyrk
dsyr2k
dtrmm
dtrsm
Routine
MKL
MLIB


MKL vs. MLIB: Improvement between sequential and parallel (2 threads) execution (%)
Dimension 10,000 x 10,000 matrices
60.00
50.00
40.00
30.00
20.00
10.00
0.00
Improvement (%)
dgemm
dsymm
dsyrk
dsyr2k
dtrmm
dtrsm
Routine
MKL
MLIB

Results

- MKL shows a better general performance in Level 1 and 2 BLAS routines and functions.
- The results in Level 3 BLAS were similar in both libraries.
- When executed in parallel, Level 3 MKL BLAS has a higher improvement than Level 3 MLIB BLAS.

Part Four

Dense AcCim performance comparison

Implementations tested

We implemented and tested the following AcCim algorithm versions for dense matrices:

Sequential versions:

- AcCim
- AcCim with MKL
- AcCim with MLIB

Parallel versions:

- AcCim MT
- AcCim MT with MKL
- AcCim MT with MLIB

Tests environment description

Hardware

Processors: 2x Intel Itanium 2

Clock frequency: 1.5 GHz

Cache: 6 MB L3

System bus bandwidth: 6.4 GB/s

Memory: 2 GB DDR, 8.5 GB/s bandwidth

Software

Operating System: Debian Linux Version 3.1r1 (Sarge)

Compiler: Intel C++ Compiler for Itanium-based applications Version 9.0.030 (20051201)

Math Libraries:

- Intel Math Kernel Library 8.0.1.006
- HP MLIB 9.2

Compiler switches used for sequential AcCim

Plain:

```
icc -O3 -fno-alias -tpp2 -mcpu=itanium2 -IPF_fma
AcCim/AcCim.cpp -o AcCim/accim
```

MKL:

```
icc -O3 -fno-alias -tpp2 -mcpu=itanium2 -IPF_fma
-D_USE_MKL_ -I/opt/intel/mkl/8.0.1/include -
L/opt/intel/mkl/8.0.1/lib/64 -lmkl AcCim/AcCim.cpp
-o AcCim/accim-mkl
```

MLIB:

```
icc -O3 -fno-alias -tpp2 -mcpu=itanium2 -IPF_fma -openmp
-D_USE_MLIB_ -I/opt/mlib/intel_8.0/hpmpi_2.1/include
-L/opt/mlib/intel_8.0/hpmpi_2.1/lib/64
/opt/mlib/intel_8.0/hpmpi_2.1/lib/64/libveclib.so.1
-lunwind
-lm /opt/mlib/intel_8.0/hpmpi_2.1/lib/64/libsolvers.so.1
AcCim/AcCim.cpp -o AcCim/accim-mlib
```

Compiler switches used for parallel AcCim

Plain:

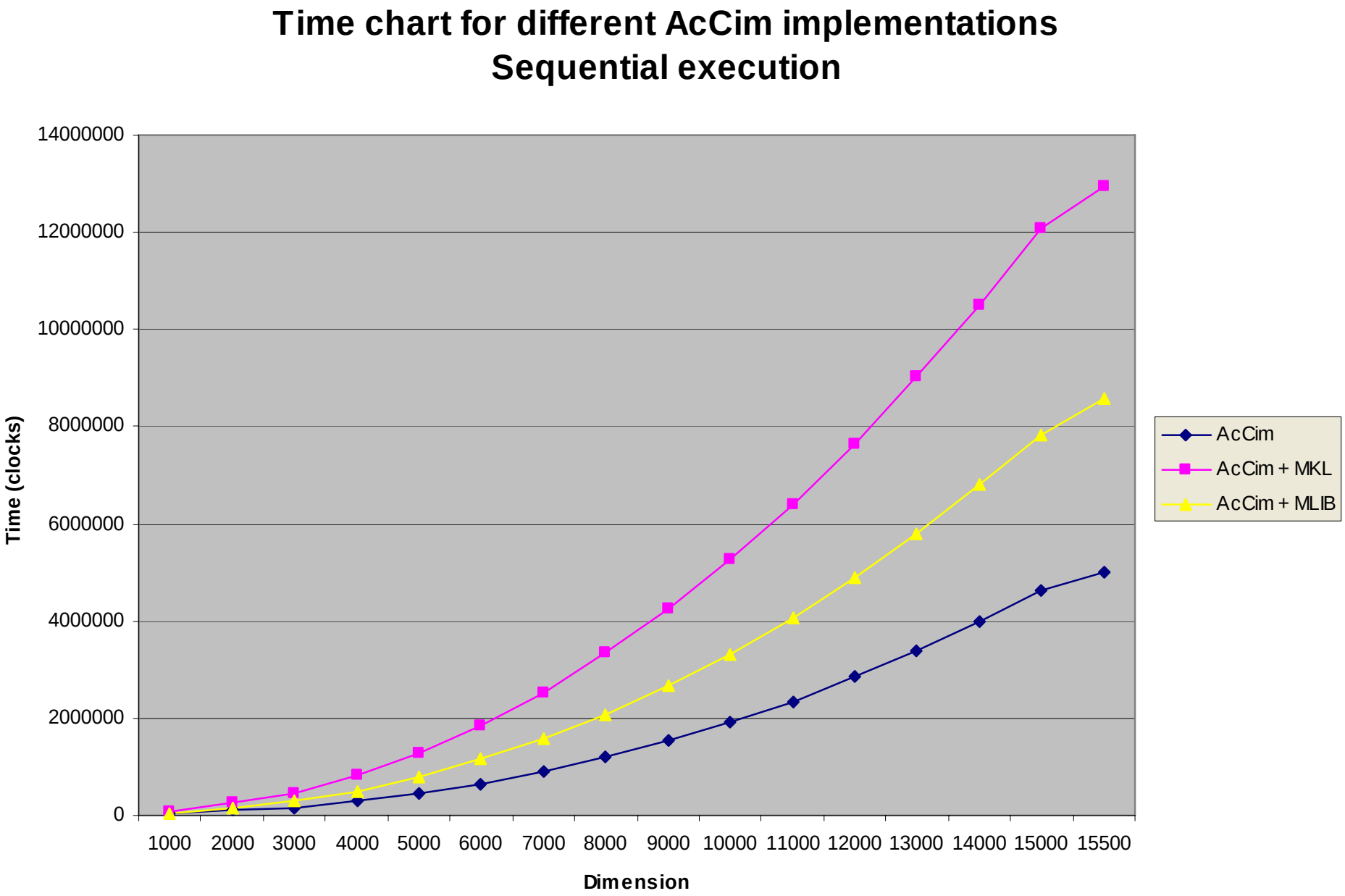
```
icc -O3 -fno-alias -tpp2 -mcpu=itanium2 -IPF_fma
-lpthread AcCimMT/AcCimMT.cpp -o AcCimMT/accimmt-d-plain
```

MKL:

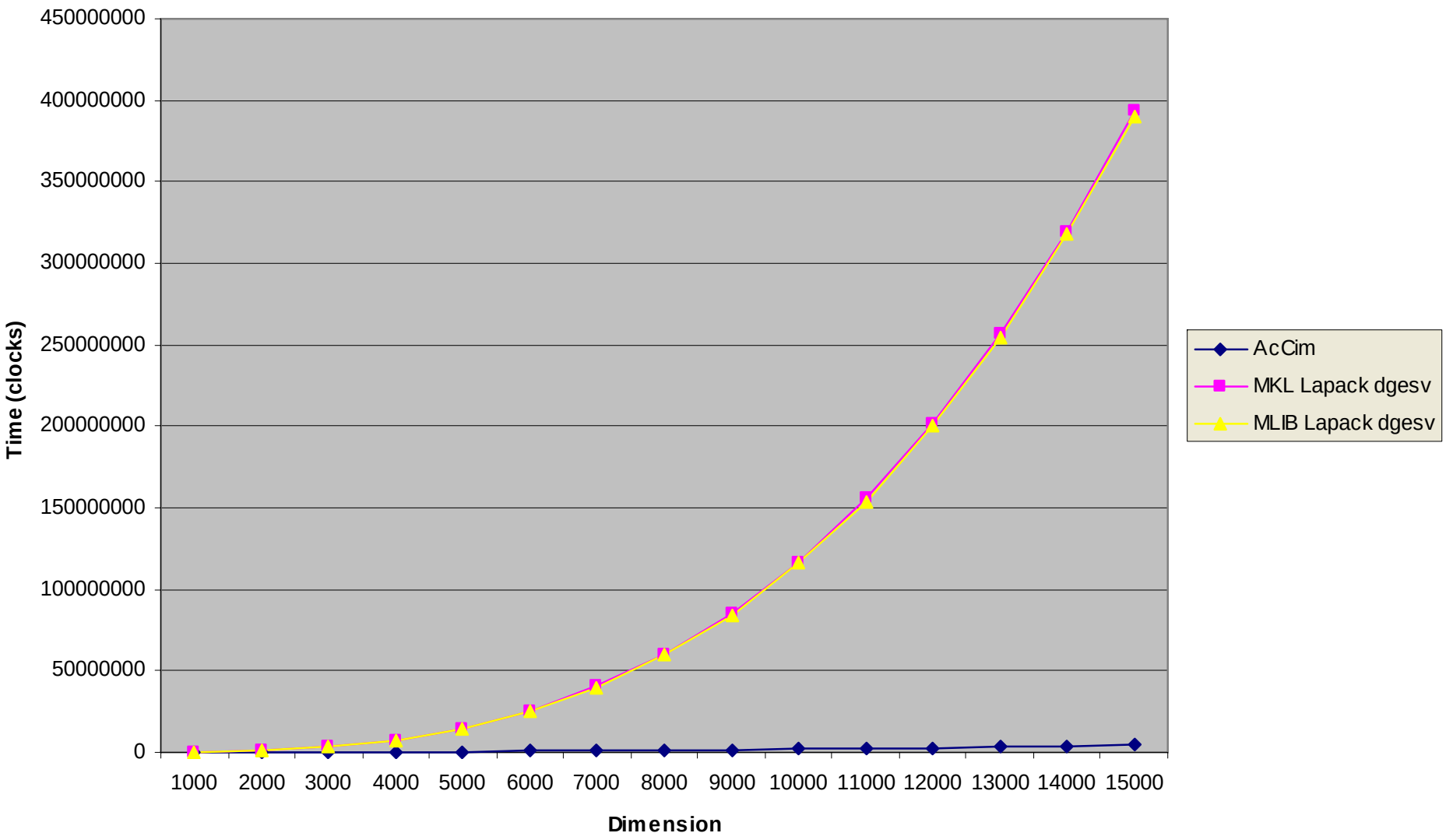
```
icc -O3 -fno-alias -tpp2 -mcpu=itanium2 -IPF_fma
-lpthread -D_USE_MKL_ -I/opt/intel/mkl/8.0.1/include
-L/opt/intel/mkl/8.0.1/lib/64 -lmkl AcCimMT/AcCimMT.cpp -o
AcCimMT/accimmt-d-mkl
```

MLIB:

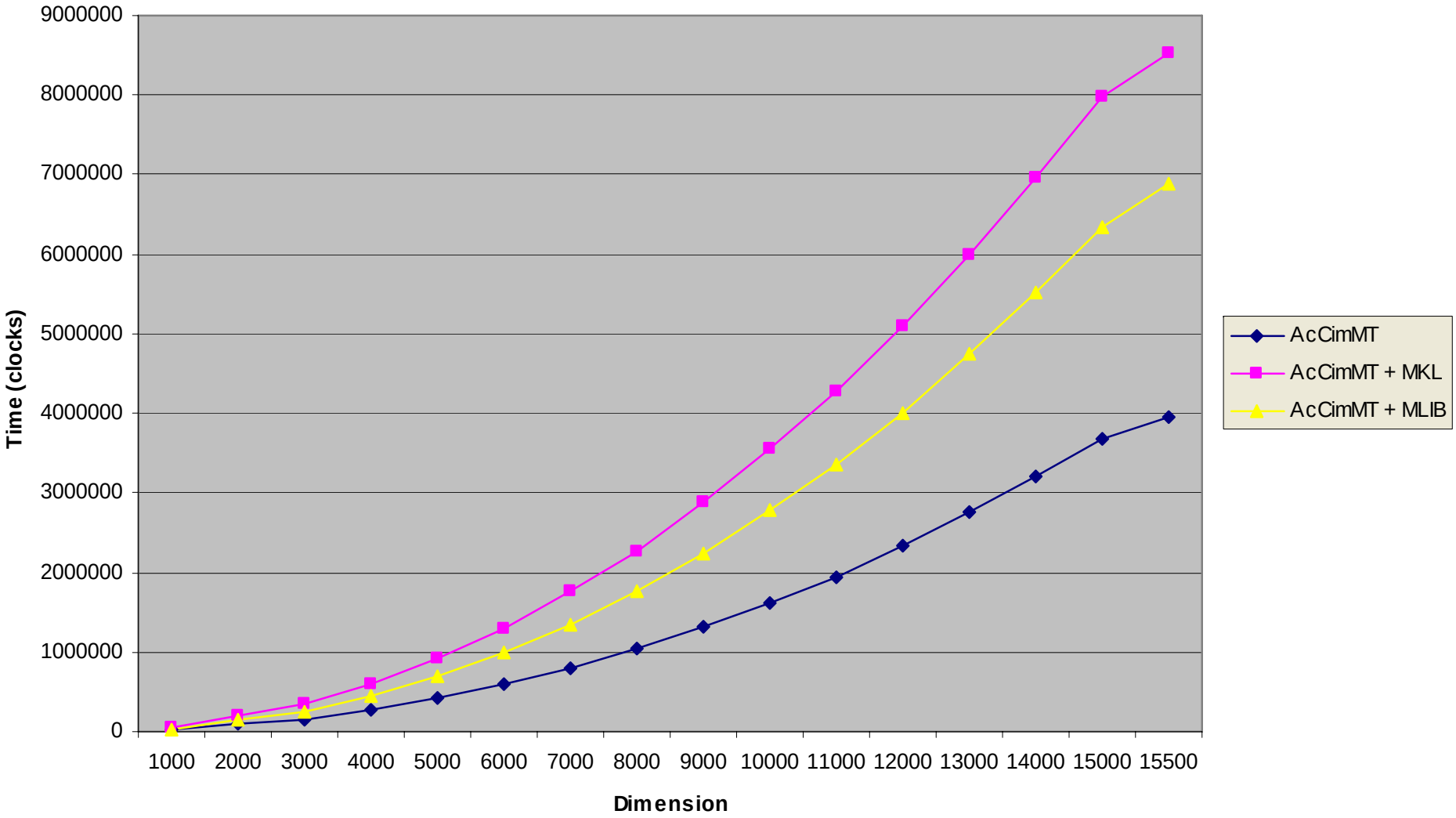
```
icc -O3 -fno-alias -tpp2 -mcpu=itanium2 -IPF_fma -openmp -
D_USE_MLIB_ -I/opt/mlib/intel_8.0/hpmpi_2.1/include
-L/opt/mlib/intel_8.0/hpmpi_2.1/lib/64
/opt/mlib/intel_8.0/hpmpi_2.1/lib/64/libveclib.so.1
-lunwind -lm
/opt/mlib/intel_8.0/hpmpi_2.1/lib/64/libsolvers.so.1
-lpthread AcCimMT/AcCimMT.cpp -o AcCimMT/accimmt-d-mlib
```


Time chart for different AcCim implementations
Sequential execution
Time (clocks)
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
15500
Dimension
AcCim
AcCim + MKL
AcCim + MLIB

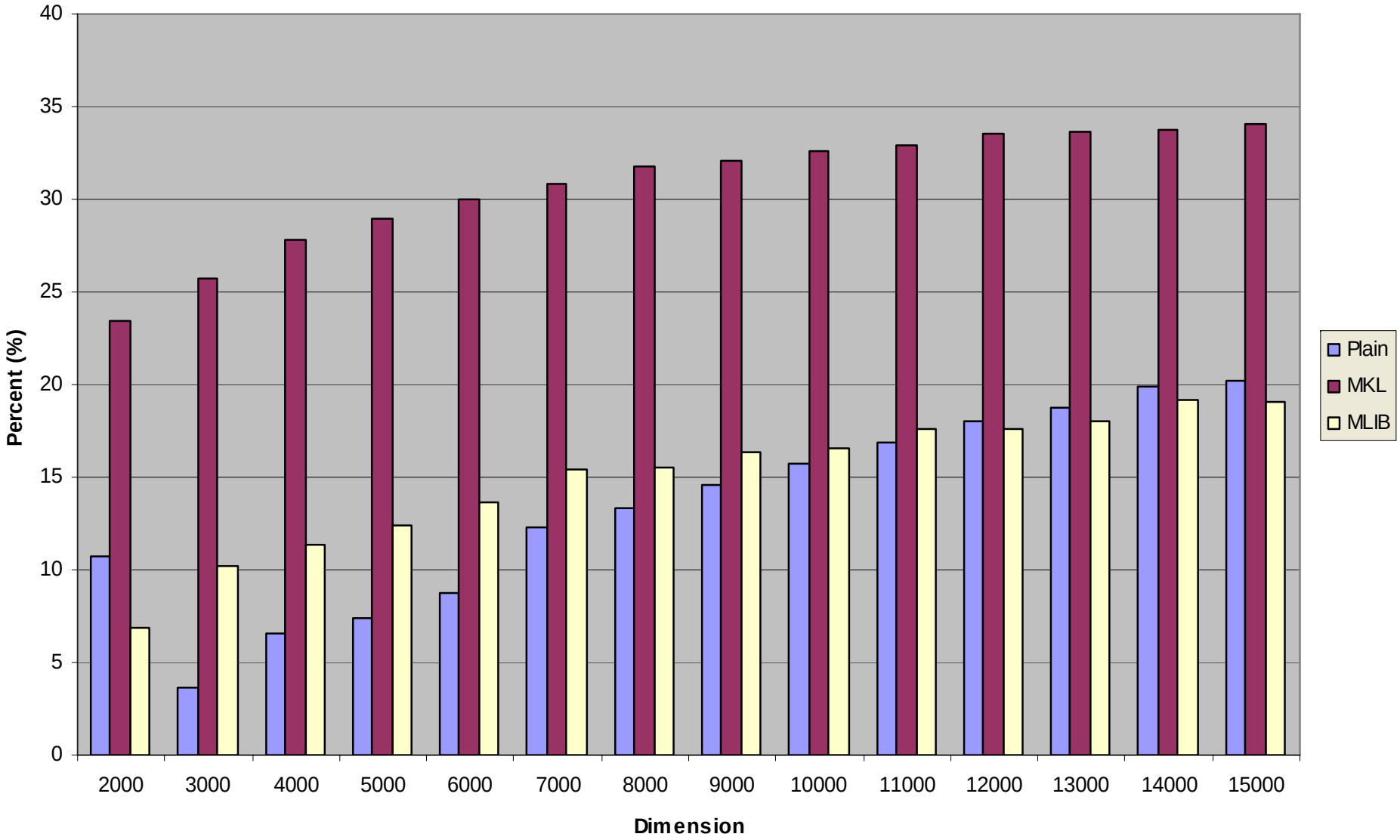
AcCim vs. LAPACK dgesv() routine



Time chart for different AcCim implementations
Parallel execution (2 threads)

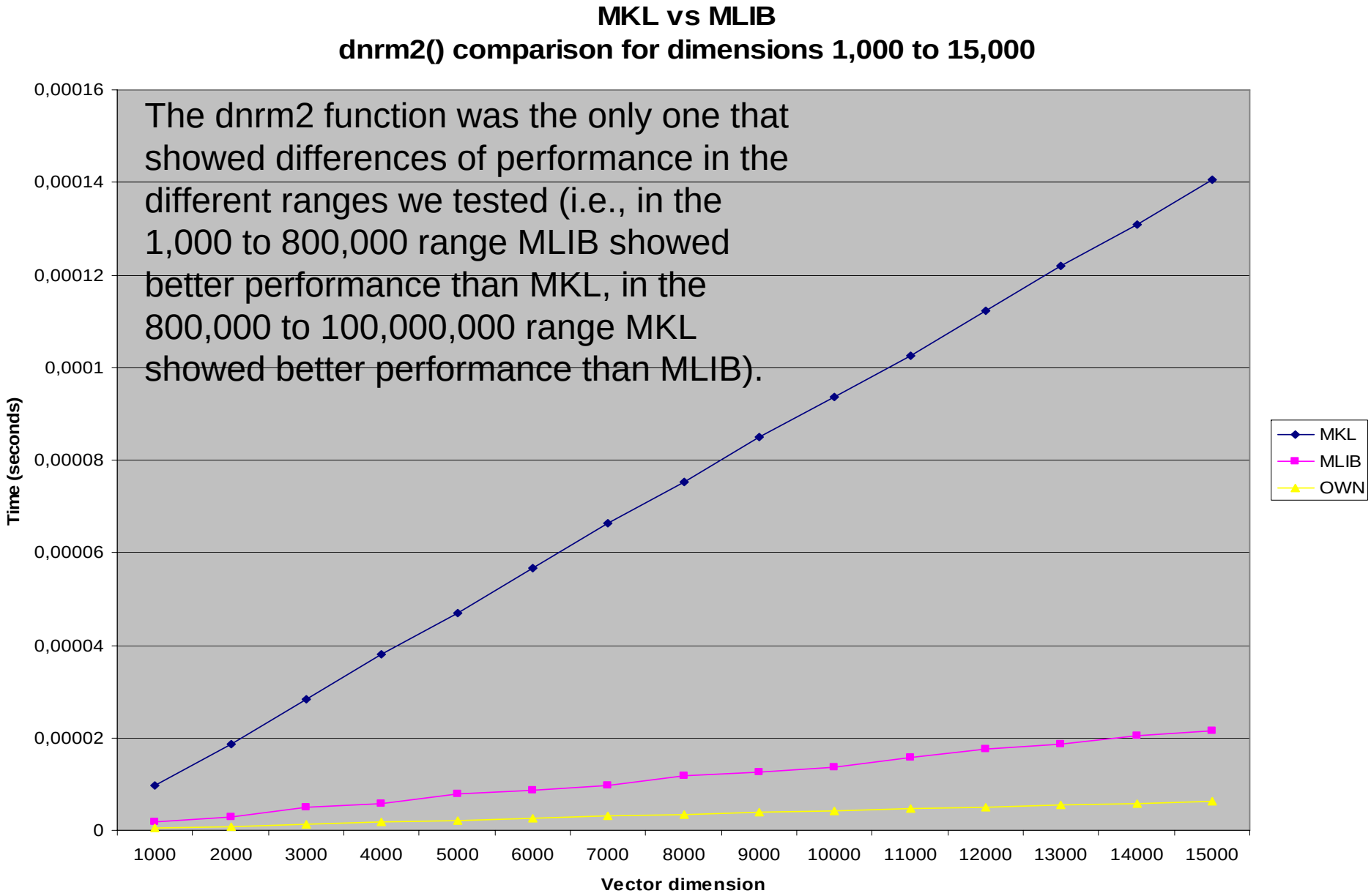


Improvement from sequential to parallel (2 threads) execution (%)



Results

- In both sequential and parallel executions we found that:
  - AcCim with our implementations of the routines and functions shows the best performance.
  - AcCim + MLIB shows better performance than AcCim + MKL.
- MKL has the highest improvement when executed in parallel.
- The results were as expected according to the performance of the most frequently used BLAS routines and functions in the AcCim algorithm for the same problem size.


MKL vs MLIB
dnrm2() comparison for dimensions 1,000 to 15,000
The dnrm2 function was the only one that showed differences of performance in the different ranges we tested (i.e., in the 1,000 to 800,000 range MLIB showed better performance than MKL, in the 800,000 to 100,000,000 range MKL showed better performance than MLIB).
Time (seconds)
0,00016
0,00014
0,00012
0,0001
0,00008
0,00006
0,00004
0,00002
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
Vector dimension
MKL
MLIB
OWN

References

- “A Class of Optimized Row Projection Methods for Solving Large Non-Symmetric Linear Systems”, H.Scolnik, N.Echebest, M.T.Guardarucci, M.C.Vacchino, Applied Numerical Mathematics 41, 4, pp.499-513, 2002, Elsevier Science. (ACCIM parallel algorithm for consistent systems)

- “New Optimized and Accelerated PAM Methods for Solving Large Non-Symmetric Systems”, H. Scolnik, N. Echebest, M.T.Guardarucci, M.C. Vacchino.in the book: Inherently Parallel Algorithms in Feasibility and Optimization and their Applications, D.Butnariu, Y. Censor and S. Reich (Editors), Studies in Computational Mathematics 8, 2001 Elsevier Science, Amsterdam, pp 457-471,2001.

- “Acceleration Scheme for Parallel Projected Aggregation Methods for Solving Large Linear Systems”, H. Scolnik, N. Echebest, M.T.Guardarucci, M.C.Vacchino. Annals of Operations Research. Volume 117, 2002, Baltzer Science Publishers.

- “An acceleration scheme for solving convex feasibility problems using incomplete projection algorithms”, N. Echebest, M.T. Guardarucci, H.D. Scolnik, M.C. Vacchino , Numerical Algorithms., 35, pp.331-350, 2004

- "An Accelerated Iterative Method with Diagonally Scaled Oblique Projections for Solving Convex Feasibility Problems", N. Echebest - M.T. Guardarucci - H. Scolnik - M.C. Vacchino, Annals of Operations Research, 138, Number 1, pp.235-257, September 2005.

- “Incomplete Oblique Projections Algorithms for Solving Large Inconsistent Linear Systems”, H. Scolnik, N. Echebest, M.T.Guardarucci, M.C. Vacchino, accepted for publication in Mathematical Programming.

Thank you!